

Using Abstraction in Multi-Rover Scheduling

Bradley J. Clement and Anthony C. Barrett

Jet Propulsion Laboratory

California Institute of Technology 4800 Oak Grove Drive, M/S 126-347

Pasadena, CA 91109-8099

{bclement, barrett}@aig.jpl.nasa.gov

Abstract

The trend toward multiple-spacecraft missions requires autonomous teams of spacecraft to coordinate their activities when sharing limited resources. This paper describes how an iterative repair planner/scheduler can reason about the activities of multiple spacecraft at abstract levels in order to greatly improve the scheduling of their use of shared resources. By finding consistent schedules at abstract levels, refinement choices can be preserved for use in robust plan execution systems. We present an algorithm for summarizing the metric resource requirements of an abstract activity based on the resource usages of its potential refinements. We find that reasoning about this summary information and that of state constraints can offer exponential improvements in the time to find consistent schedules with an iterative repair planner. We analytically describe the conditions under which these improvements are made and show that sometimes the extra overhead involved does not warrant their use. We apply these techniques within the ASPEN planner/scheduler to a domain where a team of rovers must coordinate their schedules to avoid conflicts over shared resources.

1 Introduction

Autonomous spacecraft have recently used onboard planning and execution in order to improve the efficiency of exploration by reducing explicit remote control. However, a trend toward multiple-spacecraft missions requires autonomous teams of spacecraft to collectively plan and execute for goals that arise. In order to plan for coordinated execution, the spacecraft need to reason about the population's concurrent execution to detect and resolve conflicts among the individual spacecrafts' plans. For many applications, reasoning about a planning problem at multiple levels of abstraction enhances planning and scheduling efficiency.

In an effort to reason about actions at abstract levels, Hierarchical Task Network (HTN) planners [Erol, Hendler, & Nau, 1994] represent abstract plan operators that decompose into choices of action sequences that may also be abstract.

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

These planners exploit domain knowledge to reduce the space of plans they generate. A domain expert can intuitively encode hierarchies of plan operators to guide the planner in building an ordering of actions that achieves abstract tasks, goals, and subgoals. Instead of building a plan from the beginning forward (or end backward), the planner incrementally refines abstract operators to eventually converge on specific actions that achieve the higher level goals. By structuring the refinement of goals, the planner indirectly prunes the space of inconsistent or poor plans by avoiding sequences of operators that do not effectively achieve higher level goals. While the planner is restricted to produce only those plans dictated by the structure of the hierarchy, the domain expert can still guarantee completeness by carefully structuring the abstract plan operators. The domain expert has this same responsibility when crafting operators for non-hierarchical planners.

Previous research [Korf, 1987; Yang, 1990; Knoblock, 1991] has shown that, under certain restrictions, hierarchical refinement search can reduce the search space by an exponential factor. Recent research has shown that the performance of hierarchical planners can be greatly improved without these restrictions by reasoning during refinement about the conditions embodied by abstract plan operators [Clement & Durfee, 1999; 2000]. These *summarized conditions* represent the internal and external requirements and effects of the abstract operator and those of the children in its decomposition. Using this information, a planner can resolve conflicts at abstract levels and sometimes can find abstract solutions or determine that particular decomposition choices are inconsistent. The domain expert can derive these summary conditions for each abstract operator in the domain offline before planning problems are encountered. Reasoning about this *summary information* can exponentially reduce the cost of finding a first solution and optimal solutions [Clement & Durfee, 2000].

Iterative repair planners commonly use scheduling and constraint satisfaction techniques for handling large numbers of activities and metric resources. We extend summary information to additionally include a representation for summarizing the metric resource usages of an abstract activity's decompositions. We demonstrate the benefits of abstraction in ASPEN [Chien *et al.*, 2000] (an iterative repair planner/scheduler) by using algorithms and techniques for effectively reasoning about this summary information.

While planning efficiency is a major focus of this approach, another is the support of flexible plan execution systems such as PRS [Georgeff & Lansky, 1986], UMPRS [Lee *et al.*, 1994], RAPS [Firby, 1989], JAM [Huber, 1999], etc., that

similarly exploit hierarchical plan spaces. Rather than refine abstract plan operators into a detailed end-to-end plan, however, these systems interleave refinement with execution. By postponing refinement until absolutely necessary, such systems leave themselves flexibility to choose refinements that best match current circumstances. However, this means that refinement decisions at abstract levels are made and acted upon before all of the detailed refinements need be made. If such refinements at abstract levels introduce unresolvable conflicts at detailed levels, the system ultimately gets stuck part way through a plan that cannot be completed. While backtracking is possible for HTN planning (since no actions are taken until plans are completely formed), it might not be possible when some (irreversible) plan steps have already been taken. It is therefore critical that the specifications of abstract plan operators be rich enough to summarize all of the relevant refinements to anticipate and avoid such conflicts. Using summary information, a planner can resolve conflicts at abstract levels while preserving refinement choices underneath that can be used in robust execution systems to handle unexpected or unknown events and to provide some ability to recover from failure.

This research makes the following contributions:

- An algorithm summarizing metric resource usages for abstract activities;
- Complexity analyses showing that schedule operations are exponentially cheaper at higher levels of abstraction when summarizing activities results in fewer reservations and temporal constraints;
- Experiments in a multi-rover domain that show that summary information enables a planner to find solutions (consistent schedules) more quickly under the conditions reported in the complexity analyses and that reveal conditions under which summary information can introduce unnecessary overhead;
- An empirical comparison of search techniques for directing the refinement of abstract activities based on conflicts over summarized reservations, showing how summary information can further improve performance in finding solutions.

2 Heuristic Iterative Repair

While HTN planners commonly take a generative least commitment approach to problem solving, research in the OR community illustrates that a simple local search is surprisingly effective [Papadimitriou & Steiglitz, 1998]. Planning via heuristic iterative repair involves using a local search to generate a plan. The search starts with an initial flawed plan and iteratively chooses a flaw, chooses a repair method, and changes the plan by applying the method. Unlike generative planning, the local search never backtracks. The repair methods can add, change, and remove features from the current plan. Since taking a random walk through a large space of plans is extremely inefficient, heuristics guide the choices by determining the probability distributions for each choice. We build on this approach to planning by using the ASPEN planner [Chien *et al.*, 2000].

ASPEN lets a domain expert specify two kinds of state constraints. A *must-be* constraint is a requirement that a state variable must be a specified value throughout the duration of the activity. For example, a *traffic.light* state variable may take on the values red, yellow, or green. A *cross.intersection* activity could require that

traffic.light must-be green. A *change-to* constraint changes the value of a state variable to a specified value. So, a schedule may contain periodic activities to change *traffic.light* from red to green, green to yellow, and yellow to red. Valid transitions can also be specified for state variables such that ASPEN recognizes a change from green to red as a conflict.

Resource reservations in ASPEN can be made for different types of resources. A reservation on a depletable resource has an effect that carries after the activity finishes executing. A non-depletable resource reservation only affects the resource value during the activity's execution. These reservations are usage amounts that can be positive or negative. A positive usage depletes the resource, and a negative value restores the resource. An activity places a resource reservation on a *timeline* that tracks its resultant state over the time span of the schedule. Because timeline computations are central to scheduling operations, they must be simple. The value of a *timeline unit* (a subinterval on the timeline where the value does not change) is computed assuming that resource depletion and renewal occurs at the beginning of the reservation interval. While this assumption is simplistic, it results in expedient schedule operations, and domains can still be modeled realistically by using multiple reservations to represent a more complex one and by constraining the temporal interactions of activities. A timeline unit value is the sum of all reservation values overlapping the subinterval and, in the case of a depletable resource, all downstream values from reservations of activities completed by the start of the subinterval.

Figure 1 gives an example of how reservations are made. Imagine that a rover uses battery energy for many of its tasks and can restore it using solar panels. Suppose it has a choice of two ways to perform a science experiment. It can either dig and collect a sample, or take images of a rock. In both cases, the solar panels are used to restore energy throughout the experiment. Figure 1a shows how a domain expert might model the decomposition of an *experiment* activity. Figure 1b shows how a scheduler places reservations on the battery energy timeline for two different decompositions of the *analyze* activity. The abstract *experiment* activity contains its children, and the *sunbathe* activity executes concurrently with the *analyze* activity.

ASPEN uses hierarchy to schedule groups of related activities using a technique called *aggregation* [Knight, Rabideau, & Chien, 2000]. The aggregated reservations in Figure 1c merge the individual reservations of the activities into end-to-end reservations, creating a single profile. A scheduler can then consider dropping this usage profile on the timeline in different places in order to find a suitable place in the schedule to place or move the activity hierarchy. Because the first *soak rays* activity and *dig* start at the same time, and reservation usage occurs at the beginning of the reservation interval, their usage is simply computed as the sum of -5 and 60 giving an aggregated value of 55. Again, this modeling of usage is simplistic since realistically the local usage in this interval ranges from -5 to 60. However, if this simplification potentially causes inconsistent schedules for the domain, the domain expert can easily model the activities to never start simultaneously or break up the reservations to represent more gradual usage as done with the *sunbathe* and *soak rays* activities. These aggregated profiles are only computed for activities already added to the schedule, so, in contrast to summary information, they do not represent the temporal flexi-

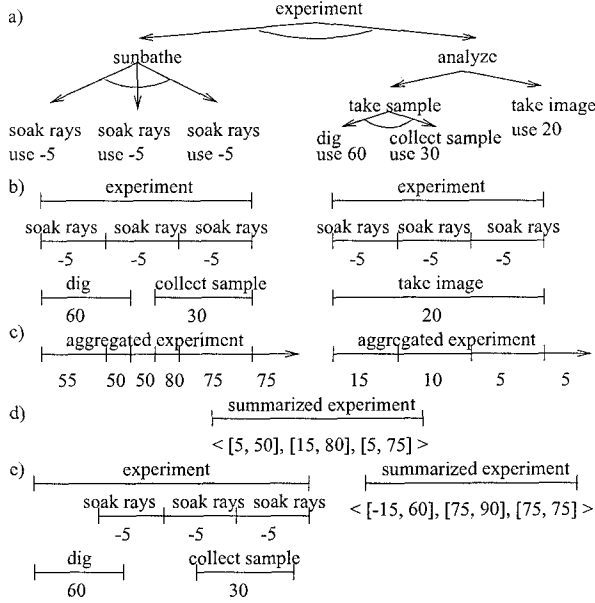


Figure 1: Resource profiles for a rover to perform a science experiment.

bility and usage ranges of reservations across decomposition choices of activities that have not been decomposed, as discussed in the next section.

3 Summarizing Reservations

As described in [Clement & Durfee, 1999], summary information captures state information for STRIPS-like plan operators [Fikes & J., 1971]. This paper describes how to extend this work for combined planning/scheduling problems. ASPEN's notation for reservations differs somewhat and additionally represents metric resource usages (see Section 2). Here we describe how we summarize this information offline for combined planner/schedulers such as ASPEN.

3.1 Summarizing State Reservations

With only a slight translation in syntax, we can use an existing algorithm to summarize state constraints. We translate activities with state reservations into CHiPs (concurrent hierarchical plans), similar to HTNs, and summarize them using the algorithm described in [Clement & Durfee, 1999]. The summary information for a CHiP includes summary pre-, in-, and postconditions. A summary precondition of a CHiP p is an externally required condition in p 's refinement that must be met by another CHiP or the initial state in order for p to execute successfully. A summary postcondition is an external effect of the CHiP's refinement (not undone internally). A summary incondition is any condition required or asserted in the CHiP's refinement within the interval of the CHiP's execution. These conditions, based on STRIPS operators, are additions, deletions, or requirements of propositional variables in the state. In addition, a summary condition is either *must* or *may* depending on whether it must hold in all or some decompositions of the plan operator. It is also either *first*, *last*, *sometimes*, or *always* indicating when in the execution interval it must hold.

We translate a must-be reservation in ASPEN into a *must*, *first* precondition and a *must*, *always* incondition for the corresponding CHiP. For a change-to reservation (described in Section 2), if it is specified to occur at the end of the activity, we translate the reservation into a *must*, *last* postcondition. If occurring at the beginning of interval, we translate the reservation into a *must*, *sometimes* incondition and a *must*, *sometimes* postcondition.¹ For example, suppose as part of the *take sample* activity (in Figure 1), a compartment for the sample is opened during the *dig* activity. The *dig* activity could have the state reservation *door* change-to *open* at end.² The *collect sample* activity could have the reservations *door* must-be *open* and *door* change-to *closed* at end. Then, after translating the reservations and summarizing *take sample*, *take sample*'s summary preconditions would be empty because the only precondition of its children is *must*, *first* (= *door open*) in *collect sample*, but it is achieved by the *dig* activity's postcondition *must*, *last* (= *door open*), so it is not an external precondition. *take sample*'s only incondition would be *must*, *sometimes* (= *door open*) because it is both an intermediate effect and requirement of the subactivities. *take sample*'s only summary postcondition would be *must*, *last* (= *door closed*) because it is an external effect that is asserted at the end of the *take sample* activity. The summary information for the *analyze* activity would include the same summary conditions as *take sample*, but all of the conditions would be *may* instead of *must* because the *take image* activity has no conditions on the compartment door, so the conditions only may need to hold for *analyze* depending on which decomposition is chosen for execution.

The state summarization algorithm [Clement & Durfee, 1999] recursively propagates summary conditions from the leaves of the hierarchy upwards. For any abstract CHiP, the algorithm derives its summary information from its immediate children with a complexity of $O(n^2c^2)$ for n activities in the hierarchy each with no more than c conditions.

3.2 Summarizing Resource Reservations

In order to extend summary information to include metric resources, we define a new representation and algorithm for summarizing metric resource usage. A *summarized resource reservation* represents ranges of values representing the potential local and downstream usage amounts of a resource. It is a tuple $\langle \text{local_min_range}, \text{local_max_range}, \text{downstream_range} \rangle$. The local usage occurs within the interval of the activity, and the downstream usage represents the depleted usage seen after the activity finishes executing. The ranges of these usages capture the multiple usage profiles of an activity that has multiple decomposition choices as well as temporal uncertainty when the domain only specifies partial orderings of activities.

Figure 1d shows how the summarized reservation represents multiple profiles for different decomposition choices. The local minimum usage ranges from 5 in the decomposition to the right in Figure 1c to 50 in the left decomposition. The local maximum usage ranges from 15 (right) to 80 (left). The downstream usages for the two decompositions are 5 (right) and 75 (left), giving the downstream range.

¹Inconditions in CHiPs were not defined to be specified as *first*.

²"at end" is a tag for specifying that the state change occurs at the end of the activity's interval.

Capturing Uncertainty in Decompositions and Temporal Ordering

So, the local min, local max, and downstream values capture changes in the usage over time (*usage profiles*), and the ranges capture values varying due to uncertainty in choices of decomposition. We represent both the minimum and maximum local ranges because, in general, a conflict could occur from violating both the minimum value and the maximum capacity. This differentiation along with usage ranges allows the planner to distinguish between potential and definite conflicts. For the example in Figure 1, if the battery energy has a capacity of 100 and is initially charged to 75, the summarized reservation in Figure 1d indicates that there might be a local conflict because the maximum local usage ranges from 15 to 80. If the battery were only charged to 10, there would definitely be a conflict. However, if fully charged to 100, there definitely would not be a conflict. This parallels the use of *must* and *may* in summary state conditions where an activity *must* or *may* clobber the conditions of another. We further discuss the use of this modal information in Section 3.4.

Now we describe how summary reservations also capture temporal uncertainty. For example, the activities in *experiment*'s refinement might not be constrained strictly as shown in Figure 1b. If the only temporal constraints were that *soak rays* activities must meet end-to-end and that *dig* must precede *collect sample*, then the execution profile could have the ordering in Figure 1e. Summary information is computed for the domain and used for abstract activities before they are refined. So, we compute resource summary reservations considering all possible orderings of the activities within the constraints specified for the domain. For the two temporal constraints that we just mentioned, the summary reservation (ignoring the *take image* branch) is computed as shown in Figure 1e. If the *sunbathe* activities precede the *take sample* activities, the minimum value of *local_min_range* is -15. The maximum of *local_min_range* is 60 because the *sunbathe* activities could follow the *dig* activity. In contrast, aggregate reservations represent only totally ordered placements of reservations with fixed start and end times.

Resource Summarization Algorithm

Like state reservations, the summarization algorithm recursively summarizes reservations from the primitive activities up the hierarchy, computing each abstract activity's reservation from the summarized reservations of its immediate children. For example, consider summarizing *p*'s reservation of some resource from the summarized reservations its children, *a*, *b*, and *c*, in Figure 2. The range values for *p*'s summarized reservation can be determined by considering the possible usage values of the children in the subintervals of *p* created by mapping the endpoints of the child activities onto the interval of *p* (like the aggregated reservations in Figure 1c). Computing *p*₁, the minimum possible usage in the combined profile, is not difficult. It is just the minimum of the children's minimum usages in the subintervals of *p*: $\min(c_1, b_1 + c_1, a_1 + b_1 + c_1, a_1 + b_1 + c_5, a_5 + b_1 + c_5)$. However, computing *p*₂, the upper bound on the minimum usage, is not so obvious. One simple algorithm would be to take the minimum of the children's maximum usages in the subintervals. This would give $p_2 = \min(c_4, b_4 + c_4, a_4 + b_4 + c_4, a_4 + b_4 + c_6, a_6 + b_4 + c_6)$. It is not obvious that this is a correct or incorrect value, but if we use the same algorithm for computing *q*₂, we have $q_2 = \min(d_4, d_6, d_6 + e_4)$.

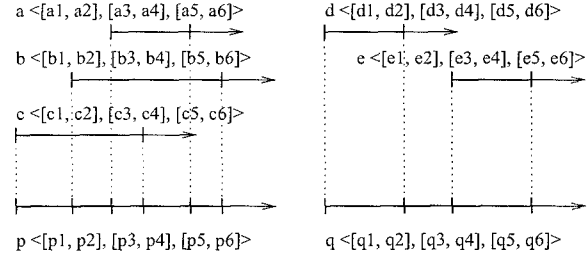


Figure 2: Summarizing the resources of two abstract activities.

This is incorrect because we know that the minimum usage in the first subinterval of *q* can be at most *d*₂, which is less than all values in the formula just given for *q*₂. *q*₂ is actually $\min(d_2, d_6, d_6 + e_2)$. The problem with the simple algorithm is that it does not consider that a minimum usage must occur in some subinterval for each activity. Thus, in computing the upper bound of the *local_min_range* (*q*₂), our algorithm (below) considers all profiles where minimum usages of the children are placed in different subintervals. The algorithm summarizes a particular abstract activity by separately processing each resource timeline affected by the parent or children. The input for the following algorithm actually includes only the activities placing a reservation on the resource timeline in question, and the calculation is based on the already summarized resource reservations of the children and any unsimplified reservation of the parent. Below, “lb” and “ub” refer to lower and upper bounds.

```

Algorithm Summarize_Resource_Timeline_Reservations
Input: parent and child activities, temporal constraints
Output: summarized resource reservation for parent
begin
  if child activities are decomposition choices
    minval ← mina ∈ activities (lb(local_min_range(a)));
    maxval ← maxa ∈ activities (ub(local_min_range(a)));
    local_min_range ← [minval, maxval];
    minval ← mina ∈ activities (lb(local_max_range(a)));
    maxval ← maxa ∈ activities (ub(local_max_range(a)));
    local_max_range ← [minval, maxval];
    minval ← mina ∈ activities (lb(downstream_range(a)));
    maxval ← maxa ∈ activities (ub(downstream_range(a)));
    downstream_range ← [minval, maxval];
    return (local_min_range, local_max_range,
            downstream_range);
  end if

  // Compute local values
  use { [x, x], [x, x], [x, x] } for the parent's
  reservation value of x;
  minmin_args, maxmin_args, minmax_args, maxmax_args ← 0;
  for each consistent sequence of interval endpoints o
    according to temporal constraints
      min_args ← 0;
      max_args ← 0;
      for all possible combinations of resource profiles
        of all activities based on their summarized
        resource reservations
          inner_args ← 0;
          for each endpoint e in o
            i ← subinterval after e and before any
            following endpoint;
            sum ← sum of reservation values
            contributed by activities overlapping i;
            insert sum into inner_args;
          end for
          insert min(inner_args) into min_args;
          insert max(inner_args) into max_args;
        end for
      insert min(min_args) into minmin_args;
      insert max(min_args) into maxmin_args;
      insert min(max_args) into minmax_args;
      insert max(max_args) into maxmax_args;
  end for

```

```

end for
local_min_range ← [min(minmin_args), max(minmax_args)];
local_max_range ← [min(maxmin_args), max(maxmax_args)];

// Compute downstream values
min_depleted ←  $\sum_{a \in \text{activities}} \min(\text{downstream\_range}(a))$ ;
max_depleted ←  $\sum_{a \in \text{activities}} \max(\text{downstream\_range}(a))$ ;
downstream_range ← [min_depleted, max_depleted];

return (local_min_range, local_max_range, downstream_range);
end

```

We represent the output reservation in terms of formulas with arguments because ASPEN reservations can be variables, with values bound to timeline values, other parameters, or user-defined functions. For example, a function of an activity's duration can determine how much battery energy is used. The formulas enable the summarization to capture complex specifications of the domain by reasoning about ranges of parameter values until they are grounded by other variables or functions in the domain. The combinations of profiles in the inner loop come from placing different range values of the summarized reservations into the subintervals (like the aggregated reservations in Figure 1c). The algorithm is exponential in the number of activities because it considers all orderings of the activities allowed by specified constraints, and exponential again in considering combinations of profiles. However, the number of immediate child activities is bounded by a constant, such that the complexity of summarizing an activity is just linear in the number of resources summarized. We do not give further complexity details here because the algorithm is offline, and the paper focuses on scheduling complexity.

As mentioned in the description of the example in Figure 2, the algorithm limits the number of reservation profiles that it considers by reasoning only about minimum and maximum reservations. Here we illustrate how the algorithm does this through an example. For the example in Figure 1, consider summarizing the battery energy usage of the *sunbathe* and *analyze* activities for *experiment*. Figure 3 shows how the upper bound of the *local_min_range* of *experiment* is computed for a particular ordering of its child activities. Figure 3a illustrates how subintervals are divided by the endpoints of *sunbathe* and *analyze*. The algorithm chooses the upper bound of the range values of the child activities' summarized reservations since they will dominate the other values in the outermost max formula. However, a local minimum reservation must occur in some subinterval. Since we are trying to find the upper bound of the local range, we can generate a profile by placing the child activity's local min upper bound in one of the subintervals and assume that the local max upper bound is placed in all other local subintervals since it will dominate all other cases in the outermost max formula. So, we must place 60 (*analyze*'s *local_min_range* upper bound) and -15 (the *local_min_range* upper bound of *sunbathe*) each in one of the two local subintervals, giving each two possible maximum profiles for mapping the local minimum and maximum into the two local subintervals as shown in Figure 3b. This means we only need to generate n profiles for the n local subintervals. Figure 3c shows the cross product of the two activities' profiles giving four possible combined profiles. The *local_min_range* upper bound is computed by taking the minimum of the reservations summed on the local subintervals for each combined profile and then taking the maximum of the result, as shown in Figure 3d. The algorithm inserts other min formulas for combined profiles of other or-

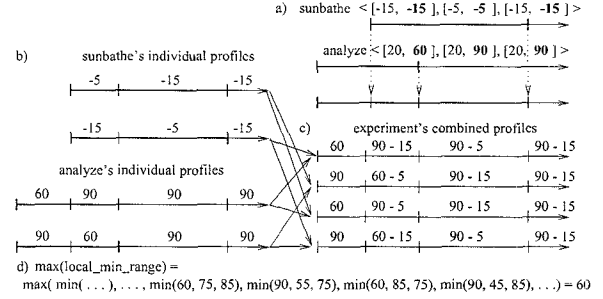


Figure 3: Deriving *experiment*'s *local_min_range* upper bound for the battery energy resource. a) The subintervals of *experiment*'s combined profile for an ordering of *sunbathe* and *analyze*. b) The possible mapping of maximum reservations to the subintervals. c) The combined profiles as a cross product of the individual profiles in 3b. d) The insertion of the summed values for the four profiles into the formula.

derings of the activities.

3.3 Computing Summarized Timelines in ASPEN

Here we describe how ASPEN uses summarized reservations to update the timeline values of resource and state variables and to detect conflicts. Timeline conflicts are often repaired by moving activities, which involves lifting their reservations from the timelines and placing them again at their new time locations. Each *timeline unit* (a subinterval between the endpoints of reservation placements) on the timeline has a distinct value. ASPEN computes these values by merging local values of overlapping reservations and propagating downstream effects of state changing and depletable resource reservations. ASPEN computes summary values for timeline units through a framework for handling user-definable state information [Knight, Rabideau, & Chien, 2001]. By only providing functions that compute timeline unit values and define conflicts, ASPEN is able to schedule activities with summarized reservations. These computations are simplifications of the summarization algorithms and are $O(1)$ when parameters are grounded.

We compute state unit values using simplifications of algorithms for summarizing CHiPs [Clement & Durfee, 1999]. We merge overlapping reservations by computing the summary information for CHiPs with the *equals* temporal relation. Likewise, we propagate downstream effects of adjacent timeline units by summarizing their corresponding CHiPs with the *meets* relation. The merge and downstream computations always act on pairs of activities, making each such computation $O(1)$. We detect a state conflict when one CHiP clobbers another, and we use ASPEN's algorithm for finding transition conflicts among possible states that *must* or *may* hold. We compute summary resource unit values similarly by summarizing activities with *equals* and *meets* temporal relations for the merge and downstream cases respectively.

These resource computations can be more complex if the values in the formulas are not yet grounded. Building the new formulas takes constant time, but determining conflicts involves computing the formulas for the ranges of the ungrounded values and is $O(p)$ for p ungrounded parameters. For each timeline unit computation, we simplify the formulas as much as possible (again $O(p)$), but delaying simplification could improve performance. The complexity analysis

and experiments in the following sections do not include this added complexity, but, domain models have generally been designed to ground all parameter values before attempting to repair conflicts, so this complexity will not be incurred for carefully designed domain models. This a subject for future research.

3.4 Using Modal Information in Summarized Reservations

In Section 3.2, we mentioned how a planner can use the modal information represented by ranges in summarized resource reservations and the existence and timing information for summarized state reservations to determine whether abstract activities *must* or *may* interact in certain ways (e.g. clobber). In this section, we discuss how backtracking and iterative repair planners use this information differently.

Backtracking planners that include ordering constraints in the search state can use this modal information to determine that an abstract schedule has no conflicts, may have some conflicts, or has some unresolvable conflicts. In the first case, the planner avoids resolving the same conflicts at lower levels in the activity hierarchies, where the complexity grows exponentially. In the latter case, the planner can backtrack in the decomposition and avoid the search space involved with trying to repair the unresolvable conflicts at the lower levels. However, iterative repair planners that do not backtrack cannot determine whether there are unresolvable conflicts, so they cannot prune the search space in the same way. But, in determining that there are definitely no conflicts, the planner does not need to differentiate between *must* and *may*. It can also ignore the upper bound of the *local_min_range* and the lower bound of the *local_max_range*—only a local min and max are needed to check whether usage exceeds the maximum capacity or falls below the minimum value. It remains unclear how a non-backtracking planner can exploit this modal information—this is a subject for future research.

If we discard the *must/may* information, a backtracking planner cannot determine unresolvable conflicts because it does not know whether the conditions in conflict will need to hold for all decompositions (*must*). However, if we discard the upper bound of the *local_min_range* and the lower bound of the *local_max_range*, a backtracking planner can still discover an unresolvable conflict among abstract activities if the minimum usages exceed the maximum capacity or the maximum usages fall below the minimum value of the resource. By also keeping track of the upper bound of the *local_min_range* and the lower bound of the *local_max_range*, the planner can discover more unresolvable conflicts because it is generally easier to discover cases when the lower bound of the *local_max_range* exceeds maximum capacity and when the upper bound of the *local_min_range* falls below the minimum value. However, there is more overhead for computing these extra two values during search, so additional research is needed to determine the effectiveness of using these values.

4 Abstract Reasoning for Iterative Repair

In this section, we describe techniques for using summary information in heuristic search to reason at abstract levels effectively and discuss the complexity advantages. Reasoning about abstract plan operators using summary information can result in exponential planning performance gains for backtracking hierarchical planners [Clement & Durfee, 2000]. In

iterative repair planning, the aggregation technique similarly outperforms the movement of activities individually [Knight, Rabideau, & Chien, 2000]. But, can summary information be used in an iterative repair planner to improve performance when aggregation is already used? We demonstrate that it makes exponential improvements by collapsing summarized reservations and temporal constraints at abstract levels. First, we analyze the complexity of moving abstract and detailed activities using aggregation. Then we describe how a heuristic iterative repair planner can exploit summary information.

4.1 Complexity of Scheduling for Aggregation and Summary Information

To move a hierarchy of activities using aggregation, valid intervals must be computed for each resource for which the hierarchy makes reservations. These valid intervals are intersected for the valid placement intervals for the abstract activity and its children. The complexity of computing the set of valid intervals for a resource is $O(rR)$ where r is the number of reservations an abstract activity makes with its children for the timeline variable, and R is the number of reservations made by other activities in the schedule on the timeline [Knight, Rabideau, & Chien, 2000]. If there are n similar activity hierarchies in the entire schedule, then $R = (n - 1)r$, and the complexity of computing valid intervals is $O(nr^2)$. But this computation is done for each of t timeline variables (often constant for a domain), so moving an activity will have a complexity of $O(tnr^2)$. The intersection of valid intervals across timelines does not increase the complexity. Its complexity is $O(tnr)$ because there can be at most nr valid intervals for each timeline; intersecting intervals for a pair of timelines is linear with the number of intervals; and only $t - 1$ pairs of timelines need to be intersected to get the intersection of the set.

The summary information of an abstract activity represents all of the reservations of its children, but if the children share reservations over the same resource, this information is collapsed into a single *summarized* reservation in the abstract activity. Therefore, when moving an abstract activity, the number of reservations involved may be far fewer depending on the domain. If the scheduler is trying to place a summarized abstract activity among other summarized activities, the computation of valid placement intervals can be greatly reduced because the r in $O(tnr^2)$ is smaller. We now consider two extreme cases where reservations can be fully collapsed and where they cannot be collapsed at all.

In the case that all activities in a hierarchy have reservations on the same timelines, the number of reservations in a hierarchy is $O(b^d)$ for a hierarchy of depth d and branching factor (number of child activities per parent) b . In aggregation, where hierarchies are fully detailed first, this means that the complexity of moving an activity is $O(tnb^{2d})$ because $r = O(b^d)$. Now consider using aggregation for moving a partially expanded hierarchy where the leaves are summarized abstract activities. If all hierarchies in the schedule are decomposed to level i , there are $O(b^i)$ activities in a hierarchy, each placing one summarized reservation representing those of all of the yet undetailed subactivities beneath it on each timeline. So $r = O(b^i)$, and the complexity of moving the activity is $O(tnb^{2i})$. Thus, moving an abstract activity using summary information can be a multiple of $O(b^{2(d-i)})$ times faster than for aggregation.

The other extreme is when all of the activities place reservations on different timelines. In this case, $r = 1$ because any hierarchy can only make one reservation per timeline. Fully detailed hierarchies contain $t = O(b^d)$ different timelines for which aggregation computes valid intervals. So, the complexity of moving an activity in this case is $O(nb^d)$. If moving a summarized abstract activity where all activities in the schedule are decomposed to level i , t is the same because the abstract activity summarizes all reservations for each subactivity in the hierarchy beneath it, and each of those reservations are on different timelines such that no reservations combine when summarized. Thus, the complexity for moving a partially expanded hierarchy is the same as for a fully expanded one. Experimental results in Section 5 exhibit great improvement for cases when activities make reservations over common resources.

Along another dimension, scheduling summarized activities is exponentially faster because it reduces the number of temporal constraints among the activities. When activity hierarchies are moved with aggregation, all of the local temporal constraints are preserved. However, there are not always valid intervals to move the entire hierarchy because the combined group of reservations can be overconstrained. (i.e. The current fixed temporal relationships in the hierarchy are not feasible.) However, the scheduler can move less constraining lower level activities to resolve the conflict. In this case, temporal constraints may be violated among the moved activity's parent and siblings. The scheduler can then move and/or adjust the durations of the parent and siblings to resolve the conflicts, but these movements can affect higher level temporal constraints or even produce other conflicts. At a depth level i in a hierarchy with decompositions branching with a factor b , the activity movement can affect b^i siblings in the worst case and produce an exponential number of conflicts. Thus, if all conflicts can be resolved at an abstract level i , $O(b^{d-i})$ scheduling operations can be avoided. In Section 5, we present empirical data showing the exponential growth of computation with respect to the depth at which ASPEN finds solutions and find many cases where summary information completed the search almost immediately because it found solutions at high levels of abstraction.

4.2 Decomposition Heuristics for Iterative Repair

Despite this optimistic complexity, reasoning about summarized reservations only translates to better performance if the movement of summarized activities resolves conflicts and advances the search toward a solution. There may be no way to resolve conflicts among abstract activities without decomposing them into more detailed activities. So when should summary information be used to reason about abstract activities, and when and how should they be decomposed? Here, we describe techniques for reasoning about summary information as abstract activities are detailed.

We explored two approaches that reason about activities from the top-level of abstraction down in the manner described in [Clement & Durfee, 2000]. Initially, the planner only reasons about the summary information of fully abstracted activities. As the planner manipulates the schedule, activities are gradually decomposed to open up new opportunities for resolving conflicts using the more detailed child activities. One strategy (that we will refer to as *level-decomposition*) is to interleave repair with decomposition as separate steps. Step 1) The planner repairs the current sched-

ule until the number of conflicts cannot be reduced. Step 2) It decomposes all abstract activities one level down and returns to Step 1. By only spending enough time at a particular level of expansion that appears effective, the planner attempts to find the highest decomposition level where solutions exist without wasting time at any level.

Another approach is to use decomposition as one of the repair methods that can be applied to a conflict so that the planner gradually decomposes activities that are involved in conflicts. This strategy tends to decompose the activities involved in greater numbers of conflicts since involved activities are potentially expanded when a conflict is repaired. The idea is that the scheduler can break overconstrained activities into smaller pieces to offer more flexibility in rooting out the conflicts. This resembles the EMTF (expand-most-threats-first) [Clement & Durfee, 2000] heuristic that expands (decomposes) activities involved in conflicts before others. (Thus, we will refer to this heuristic as EMTF throughout the rest of this paper.) Activities that are not involved in conflicts are rarely expanded because they are less likely chosen for repair. Experiments in Section 5 suggest that EMTF performs better than level-decomposition, but only when EMTF uses decomposition rates suited for the problem domain.

Another heuristic for improving planning performance prefers decomposition choices that lead to fewer conflicts. Using summary information, the planner can test each child activity by decomposing to the child and replacing the parent's summarized reservations that summarize the children with the particular child's summarized reservations. For each child, the number of conflicts in the schedule are counted, and the child creating the fewest conflicts is chosen.³ This is the *fewest-threats-first* (FTF) heuristic that was demonstrated to be very effective in pruning the search space in a backtracking planner [Clement & Durfee, 2000]. Consistently, the experiments in Section 5 report that using FTF can find solutions much more quickly but only when decomposition choices cause significantly varying numbers of conflicts.

5 Multi-Rover Domain Experiments

The experiments we describe here show that, for our chosen domain, summary information improves performance significantly when activities within the same hierarchy make reservations over the same resource, and solutions at some level of abstraction are found. At the same time, we find cases where reasoning at abstract levels incurs significant overhead when solutions are only found at deeper levels. However, in domains where decomposition choices are critical, we show that this overhead is insignificant because the FTF heuristic finds solutions at deeper levels with better performance. These experiments also show that the EMTF heuristic outperforms level-decomposition for certain decomposition rates, raising new research questions. In addition, we show that the time to find a solution increases dramatically with the depth where solutions are found, supporting the notion that more constraints at deeper levels exponentially complicates the scheduling problem.

Our problems consist of a team of rovers that must resolve conflicts over shared resources. We generate two classes of maps within which the rovers move. For one, we randomly

³Or, in stochastic planners like ASPEN, the children are chosen with probability decreasing with their respective number of conflicts.

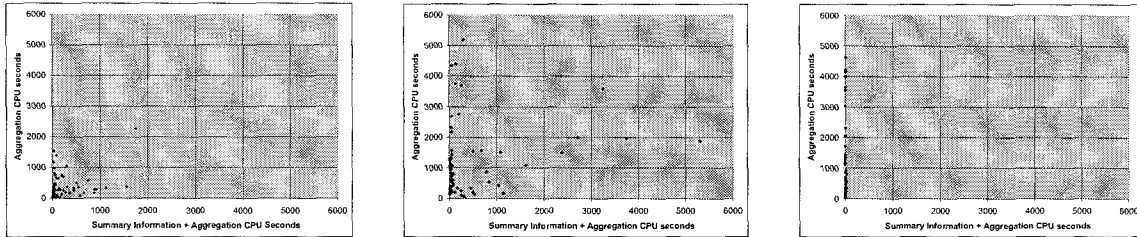


Figure 4: Plots for the *no channel*, *mixed*, and *channel only* domains

generate a map by placing paths among random waypoints in a field using a Delaunay triangulation algorithm. For the other, we generate corridor paths from a circle of locations with three paths from the center to points on the circle to represent narrow paths around obstacles. This latter map is used only for an experiment evaluating the FTF heuristic. We then select a subset of the points as science locations and use a simple multiple traveling salesman algorithm to assign routes for the rovers to traverse and perform experiments. Paths between waypoints are assigned random capacities such that either one, two, or three rovers can traverse a path simultaneously; only one rover can be at any waypoint; and rovers may not traverse paths in opposite directions. In addition, rovers must communicate with the lander for telemetry using a shared channel of fixed bandwidth. Depending on the terrain between waypoints, the required bandwidth varies. 80 problems were generated for two to five rovers, three to six science locations per rover, and 9 to 105 waypoints. In general problems that contain fewer waypoints and more science locations are more difficult because there are more interactions among the rovers. Schedules ranged in size from 180 to 1300 activities. Note that the experiments use a prototype interface in order to use summary information, and some of ASPEN's optimized scheduling techniques could not be used. However, we report relative performance, making the comparisons fair.

Schedules consist of an abstract activity for each rover that decomposes into activities for visiting each assigned science location. Those activities decompose into the three shortest paths through the waypoints to the target science location. The paths decompose into movements between waypoints. Additional levels of hierarchy were introduced for longer paths in order to keep the offline resource summarization tractable.

We compare the use of iterative repair with and without summarization in the context of aggregation for three variations of the triangulated field domain. The use of summary information includes the EMTF and FTF heuristics for decomposition. One domain excludes the communications channel resource (*no channel*); one excludes the path capacity restrictions (*channel only*); and the other includes all mentioned resources (*mixed*). Since all of the movement activities reserve the channel resource, we expect greater improvement in performance when using summary information according to the complexity analyses in the previous section. Activities within a rover's hierarchy rarely place reservations on other timelines more than once, so the *no channel* domain corresponds to the case where summarization collapses no reservations.

Figure 4 (left) exhibits two distributions of problems for the *no channel* domain. In most of the cases (points along the

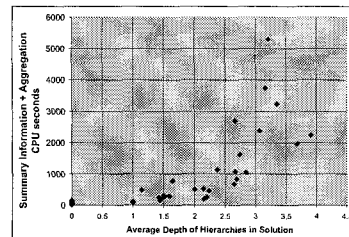


Figure 5: CPU time for solutions found at varying depths.

y-axis), ASPEN with summary information finds a solution quickly at some level of abstraction. However, in many cases, summary information performs notably worse (points along the x-axis). We find that for these problems finding a solution required digging deep into the rovers' hierarchies, and once it decomposes the hierarchies to these levels, the difference in the additional time to find a solution between the two approaches is negligible unless the use of summary information found a solution at a slightly higher level of abstraction more quickly. Thus, the time spent reasoning about summary information at higher levels incurred unnecessary overhead. Previous work shows that this overhead is rarely significant in backtracking planners because summary information can prune inconsistent search spaces at abstract levels [Clement & Durfee, 2000]. However, in non-backtracking planners like ASPEN, the only opportunity we found to prune the search space at abstract levels was using the FTF heuristic to avoid greater numbers of conflicts in particular branches. Later, we will explain why FTF is not helpful in the triangulated field domains, but is very effective in the corridor domain.

Figure 4 (middle) shows significant improvement for summary information in the *mixed* domain compared to the *no channel* domain. Adding the channel resource rarely affected the use of summary information because the collapse in summary reservations incurred insignificant complexity on top of the other reservations. However, the channel resource made the scheduling task noticeably more difficult for ASPEN when not using summary information. In the *channel only* domain (Figure 4 right), summary information finds solutions at the abstract level almost immediately, but the problems are still complicated when ASPEN does not use summary information. These results support the complexity analysis in the previous section that argues that summary information exponentially improves performance when activities within the same hierarchy make reservations over the same resource and solutions are found at some level of abstraction.

Figure 5 shows the CPU time required for ASPEN using summary information for the *mixed* domain for the depths

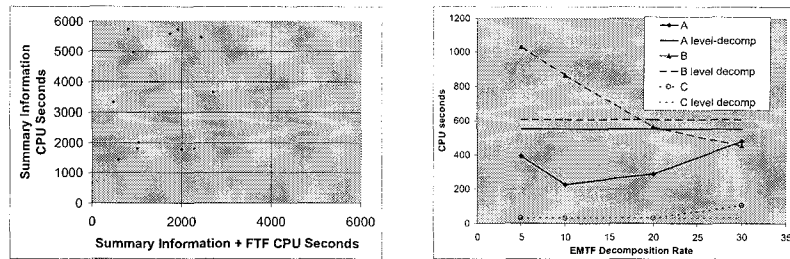


Figure 6: Performance using FTF and EMTF vs. level-decomposition heuristics.

at which the solutions are found. The depths are average depths of leaf activities in partially expanded hierarchies. The CPU time increases dramatically for solutions found at greater depths, supporting our claim that finding a solution at more abstract levels is exponentially easier.

For the triangulated field domain, choosing different paths to a science location usually did not make a significant difference in the number of conflicts encountered because if the rovers crossed paths, all path choices would still lead to conflict. In the corridor domain, however, path choices would always lead down a different corridor to get to the target science location, so there was usually a path that would avoid a conflict and a path that would cause one. Using the FTF heuristic dominated the planner choosing decompositions randomly for all but two problems (Figure 6 left).

Figure 6 (right) shows the performance of EMTF vs. level decomposition for different rates of decomposition for three problems selected from the set. The plotted points are averages over ten runs for each problem. Depending on the choice of rate of decomposition (the probability that an activity will decompose when a conflict is encountered), performance varies significantly. However, the best decomposition rate can vary from problem to problem making it difficult for the domain expert to choose. Our future work will include investigating the relation of decomposition rates to performance based on problem structure.⁴

6 Conclusion

Reasoning about abstract resource reservations exponentially accelerates finding schedules when reservations collapse during summarization, and abstract solutions can be found. Similar speedups occur when decomposition branches result in varied numbers of conflicts. The offline algorithm for summarizing metric resource usage makes these performance gains available for a larger set of expressive planners and schedulers. We have shown how these performance advantages can improve ASPEN's effectiveness when scheduling the activities of multiple spacecraft. The use of summary information also enables a planner to preserve decomposition choices that robust execution systems can use to handle some degree of uncertainty and failure. We have begun to apply these techniques to a domain where a collection of satellites, each with a set of sensors, must schedule individual and team measurements to meet the conflicting agendas of a group of scientists. We are also interested in developing protocols to allow multiple spacecraft planners to coordinate their activities asynchronously during execution.

⁴For other experiments, we used a decomposition rate of 20%.

References

- [Chien *et al.*, 2000] Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*.
- [Clement & Durfee, 1999] Clement, B., and Durfee, E. 1999. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*.
- [Clement & Durfee, 2000] Clement, B., and Durfee, E. 2000. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*.
- [Erol, Hendler, & Nau, 1994] Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.
- [Fikes & J., 1971] Fikes, R. E., and J., N. N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.
- [Firby, 1989] Firby, J. 1989. *Adaptive Execution in Complex Dynamic Domains*. Ph.D. Dissertation, Yale University.
- [Georgeff & Lansky, 1986] Georgeff, M. P., and Lansky, A. 1986. Procedural knowledge. *Proc. IEEE* 74(10):1383–1398.
- [Huber, 1999] Huber, M. 1999. Jam: a bdi-theoretic mobile agent architecture. In *Proc. Intl. Conf. Autonomous Agents*, 236–243.
- [Knight, Rabideau, & Chien, 2000] Knight, R.; Rabideau, G.; and Chien, S. 2000. Computing valid intervals for collections of activities with shared states and resources. In *Proc. AIPS*, 600–610.
- [Knight, Rabideau, & Chien, 2001] Knight, R.; Rabideau, G.; and Chien, S. 2001. Extending the representational power of model-based systems using generalized timelines. ISAIRAS (abstract submitted).
- [Knoblock, 1991] Knoblock, C. 1991. Search reduction in hierarchical problem solving. In *Proc. AAAI*, 686–691.
- [Korf, 1987] Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.
- [Lee *et al.*, 1994] Lee, J.; Huber, M. J.; Durfee, E. H.; and Kenny, P. G. 1994. Umprs: An implementation of the procedural reasoning system for multirobot applications. In *Proc. AIAA/NASA Conf. on Intelligent Robotics in Field, Factory, Service, and Space*, 842–849.
- [Papadimitriou & Steiglitz, 1998] Papadimitriou, and Steiglitz. 1998. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications New York.
- [Yang, 1990] Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence* 6(1):12–24.